IN THE CLAIMS:

Please find below a listing of all of the pending claims. The statuses of the claims are set forth in parentheses.

1. (Previously Presented) A method for testing software, comprising:

examining an application software program including calls to system classes with both a static analysis tool and a dynamic analysis tool;

determining a static use count of said system classes from the examining;

deriving a dynamic use count of each of said system classes during operation of said application software program from the examining;

assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count; and

testing said system classes in order according to said corresponding proportional weighing attributes.

2. (Original) The method of claim 1, wherein:

the step of testing is such that only the most heavily weighted portion of all such system classes are tested at all.

3. (Original) The method of claim 1, wherein:

the step of testing is such that only those system classes that are actually used in operation of said application software program are tested at all;

wherein, costs and delays associated with such pointless testing are avoided.

4. (Original) The method of claim 1, wherein:

producing a static use count further comprises assigning a static observation percentage to each system class by dividing said static use count by a sum of all static use counts.

5. (Original) The method of claim 1, wherein:

producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing said dynamic use count by a sum of all dynamic use counts.

6. (Original) The method of claim 1, wherein:

producing a static use count further comprises assigning a static observation percentage to each system class by dividing the static use count by a sum of all static use counts; and

producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing the dynamic use count by a sum of all dynamic use counts.

7. (Previously Presented) The method of claim 6, wherein the step of assigning to each of the system classes a weight based on the static use count and the dynamic use count further comprises the steps of:

assigning to a public untested system class in the system classes a first weight defined by a first constant plus a sum of the static use count plus the dynamic use count;

assigning a private untested software class in the system classes a second weight that is equal to the first constant;

assigning to each public function in the system classes that is not fully tested a third weight that is defined as a second constant that is less than the first constant, to which is added a sum of the static observation percentage plus the dynamic observation percentage; and

assigning to all remaining public and private functions in the system classes a fourth weight defined as a third constant that is less than the second constant.

8. (Original) The method of claim 1, wherein:

the testing the system classes further comprises ending a test when a testing resource is exhausted and prior to testing a last entry having a least weight.

9. (Original) The method of claim 8, wherein:

the testing the system classes further comprises ending a test when at least a limit of available time or funding is exhausted and prior to testing a last entry having a least weight.

10. (Previously Presented) A machine-readable medium on which is encoded machine-readable code for testing object-oriented system software having system classes, the machine readable code comprising:

machine-readable code for running a static analysis tool for examining an application software program, the application software program including calls to the system classes;

machine-readable code for determining a static use count of the system classes in the application software program from the result;

machine-readable code for running a dynamic analysis tool for examining the application software program and producing a dynamic use count based on the application software program's dynamic use of the system functions while running the application software program;

machine-readable code for assigning to each system class a weight based on the static use count and the dynamic use count, and

machine-readable code for testing the system classes, in order, based on the assigned weight, from a first entry having a greatest weight.

11. (Original) The software of claim 10, wherein:

producing a static use count further comprises assigning a static observation percentage to each system class by dividing the static use count by a sum of all static use counts.

12. (Original) The software of claim 10, wherein:

producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing the dynamic use count by a sum of all dynamic use counts.

13. (Original) The software of claim 10, wherein:

producing a static use count further comprises assigning a static observation percentage to each system class by dividing the static use count by a sum of all static use counts, and

producing a dynamic use count further comprises assigning a dynamic observation percentage to each system class by dividing the dynamic use count by a sum of all dynamic use counts.

14. (Previously Presented) The software of claim 10, wherein the assigning to each of the system classes a weight based on the static use count and the dynamic use count further comprises the steps of:

assigning to a public untested system class in the system classes a first weight defined by a first constant plus a sum of the static use count plus the dynamic use count;

assigning a private untested software class in the system classes a second weight that is equal to the first constant;

assigning to each public function in the system classes that is not fully tested a third weight that is defines as a second constant that is less than the first constant, to which is added a sum of the static observation percentage plus the dynamic observation percentage; and

assigning to all remaining public and private functions in the system classes a fourth weight defined as a third constant that is less than the second constant.

15. (Previously Presented) A machine-readable medium on which is encoded a software tester program code, the software tester program code comprising:

means for examining an application software program including calls to system classes with both a static analysis tool and a dynamic analysis tool;

means for determining a static use count of said system classes from the examining;

means for deriving a dynamic use count of each of said system classes during operation of said application software program from the examining;

means for assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count; and

means for testing said system classes in order according to said corresponding proportional weighing attributes.

16. (Original) The tester of claim 15, wherein:

the means for testing is such that only the most heavily weighted portion of all such system classes are tested at all.

17. (Previously Presented) A business model for testing software, comprising:

setting a resource limit on the available time or money that is devoted to testing a particular application software program;

examining said application software program including calls to system classes with both a static analysis tool and a dynamic analysis tool;

determining a static use count of said system classes from the examining;

deriving a dynamic use count of each of said system classes during operation of said application software program from the examining;

assigning a proportional weighing attribute to each system class based on its corresponding static use count and dynamic use count;

testing said system classes in order according to said corresponding proportional weighing attributes and proceeding down to the least heavily weighted system classes; and

stopping testing when said resource limit is reached.